• E-Mail: practicalwireless@warnersgroup.co.uk

WSJT-X Version 2

Mike Richards G4WNC has the latest on WSJT-X, an aside on digital beacons and some insights into the future of SDR.

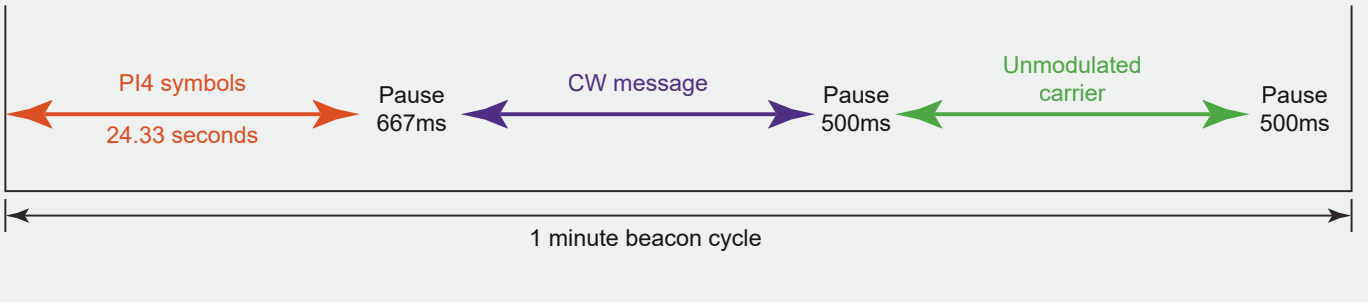

Fig. 1: Transmission sequence for PI4 beacons.

First thing to note this month is the full release of WSJT-X version 2 on December 10th, 2018. This is a mandatory update for all users because both the message length and CRC (Cyclic Redundancy Checks) have changed and there is no backward compatibility. If you don't upgrade, you will be unable to make contacts using FT8 or MSK144 modes.

As mentioned in previous months, the changes bring several important improvements to both modes that are especially helpful for contest work. Rather than repeat the details here, you can see a full list of the changes on the WSJT-X website at:
https://tinyurl.com/yafb6wgh

The latest version of WSJT-X can also be downloaded from:
https://tinyurl.com/hg6rnxm

For those of you that have purchased my ready-programmed Raspberry Pi SD cards, here's a step-by-step update guide for version 2. You can also use these instructions to do a fresh install on a Raspberry Pi 3B running the standard Raspbian Stretch distribution:

1. Open a browser and navigate to:
https://tinyurl.com/hg6rnxm
2. Scroll to the WSJT-X 2.0 section at the bottom of the page and download the Raspbian ARMv6 file. It will probably be called: **wsjtx_2.0.0_armhf.deb**
3. Open a Terminal session
4. Enter: **cd ~Downloads** to switch to the Downloads folder
5. When in the Downloads folder, enter: **sudo dpkg -i wsjtx_2.0.0_armhf.deb** (or the name of the file you downloaded, if different). This will unpack and install WSJT-X
6. Close the Terminal session and you're done!

As you can see, it's a simple process and all your settings and station details should be preserved. Operating the new version is slightly different, especially for contest work, so I recommend reading the *Quick-Start Guide* to familiarise yourself with the changes. You can find the *Quick-Start Guide* here:
https://tinyurl.com/yam7xy9a

If you want to purchase one of my ready programmed *Data Modes* cards, you can visit my shop at:
http://g4wnc.com/shop-2

I've recently improved my offering and now supply, fast, 16GB, Class 10 microSD cards with no price increase! In addition to the latest versions of WSJT-X, I include JS8Call, FLDIGI and QSSTV.

Digital Beacons

One of the many interesting talks at this year's RSGB Convention was **Bo Hansen OZ2M's** lecture on the use of digital modes for VHF beacons. After much experimentation, PI4 (PharusIgnis4) has been selected as the most appropriate data mode and has already been implemented on many European beacons. The name is derived from the ancient Greek for Lighthouse (Pharus) and Latin word for fire (Ignis), while the number 4 relates to the 4-tone modulation. PI4 mode is ideal for beacons because it complies with the IARU Region 1 VHF Committee requirements and offers a Signal-to-Noise performance of between –22dB and –23dB, which is close to that achieved by JT4. These PI4 beacons use a one-minute, mixed-mode, repeating, transmit sequence that comprises PI4 data then Morse followed by an unmodulated carrier. I've illustrated the sequence illustrated in **Fig. 1**. As you can see, this facilitates easy detection by ear using Morse but also enables high sensitivity digital decoding that should help resolve very weak beacon signals. You will need dedicated software to receive the PI4 data and there are a few free options to choose from. For Windows users, PI-RX, **Fig. 2**, is probably your best bet, and can be downloaded from:
http://rudius.net/oz2m/software/pi-rx

There is another package, called MSHV and developed by **Christo LZ2HV**, that includes both Linux and Windows versions and is available from:
https://tinyurl.com/yahsp5pd

MSHV is particularly interesting because it is an adaptation if the WSJT-X suite of software so includes the full range of weak signal modes plus the PI4 beacon.

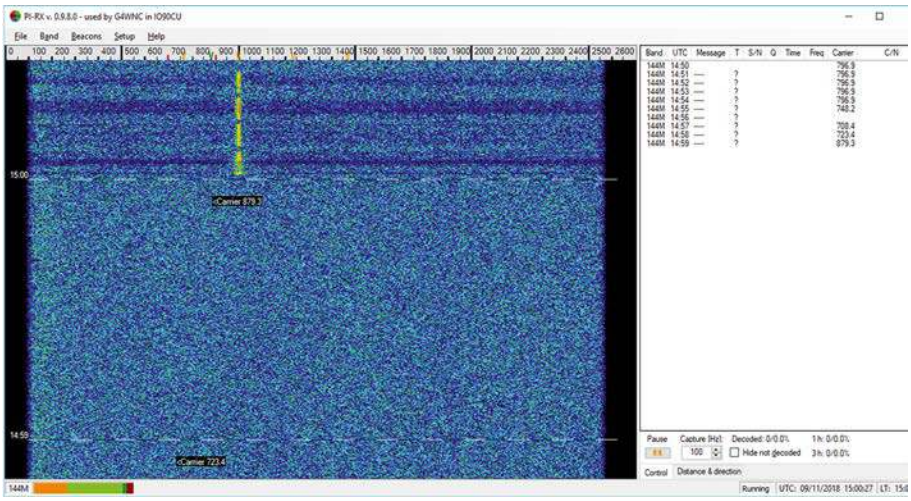

Fig. 2: PI-RX Windows software for decoding PI4 beacons.

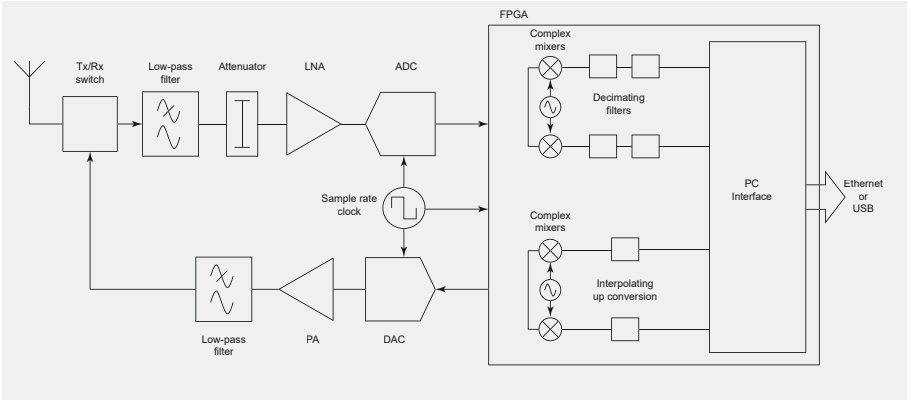

Fig. 3: Present day high-end SDR hardware.

Future of SDR?

At the 2018 Friedrichshafen Ham Radio event, **Phil Harman VK6PH** gave a very interesting lecture on an alternative approach to high-end SDRs known as Direct Fourier Conversion (DFC). You can see the full lecture on YouTube (URL below) but I think it's worth covering here because it contains some important new ideas.
https://tinyurl.com/ycxxeyt3

Over the last few years, the top players in the world of amateur radio SDR seem to have settled on a transceiver architecture that puts the ADC (Analogue to Digital Converter) and DAC (Digital to Analogue Converter) as close as possible to the antenna. This makes good sense and gets the entire HF band in and out of digital format with minimal analogue processing. The problem here is that the data rates of the ADC and DAC are huge. The better designs tend to use 16-bit ADCs and sample rates around the 120MSPS rate, to provide coverage from LF through to the 50MHz band. That leaves us dealing with 16-bit data that's arriving 120 million times per second, that's just under 2Gb/s of data! In addition to handling the data flow, work needs to be done on that data, all of which requires fast processing channels.

The transmit chain has a similar problem because the DAC needs to be fed with 16-bit data arriving at 122MSPS. The standard way to handle this is to use an FPGA (Field Programmable Gate Array) immediately after the ADC and before the DAC to perform the DDC/DUC (Digital Down/Up Conversion). FPGAs are used because they have a huge number of configurable, high-speed, logic blocks that can be linked together using programmable interconnects. The result is a complex processing unit that uses parallel programming to make light work of the high-speed data. The FPGAs also have the advantage of being software configurable, so you can completely change the features of the SDR simply by uploading a revised set of interconnects.

Creating the down-sampled output from the FPGA requires the use of a significant number of logic blocks and I've illustrated a typical SDR configuration in **Fig. 3**. Here you can see that the incoming data is initially fed to a pair of digital mixers that have a quadrature (90° out of phase) local oscillator. This is required to produce the complex I and Q signals we need to support the digital filtering and demodulation. Each of the mixer outputs is passed to a digital lowpass filter and then a decimator to provide a lower sample-rate IQ stream that can be passed over USB or Ethernet to the host computer. However, as we have seen with some recent rigs, it's quite possible to complete the digital processing entirely within the rig, so you no longer need a computer to run the SDR. While this solution will suit some, it does limit the flexibility benefits of SDR technology.

As you can see from Fig. 3, there is significant work to be done in the FPGA, which translates to higher hardware costs and partly explains why this technology remains expensive. In addition to the hardware, FPGA programming is a very specialised skill, so labour is also expensive. Of course, if you want to include multiple receivers, that means more logic blocks, hence bigger and more expensive FPGAs. One alternative that could avoid the FPGA altogether would be to take advantage of the new PC communication technologies such as PCIe where data rates of up to 12Gb/s can be achieved. All the FPGA work could then be done by the PC. However, this would require a top-end PC to process the data and there wouldn't be much processor capacity to do anything else!

A Different Solution

In his lecture, Phil Harman presented a different solution that completely changes the way we deal with the ADC data. The present approach does most of the processing in the time domain but Phil's alternative is to process in the frequency domain. Swapping between time and frequency domains is quite common in DSP (Digital Signal Processing) and a good example can be found in the detailed spectrum and waterfall displays that are a standard feature of modern SDRs. Converting a signal from the time to frequency domain is normally done with an FFT (Fast Fourier Transform). The FFT can take an incoming stream of data samples and convert this into a series of memory locations, each of which contains a number that represents the signal power of a specific, narrow, frequency band. These memory locations are commonly called bins and the size of each bin is determined by: sample-rate/number of bins. I've shown an illustration of an FFT in **Fig. 4**.

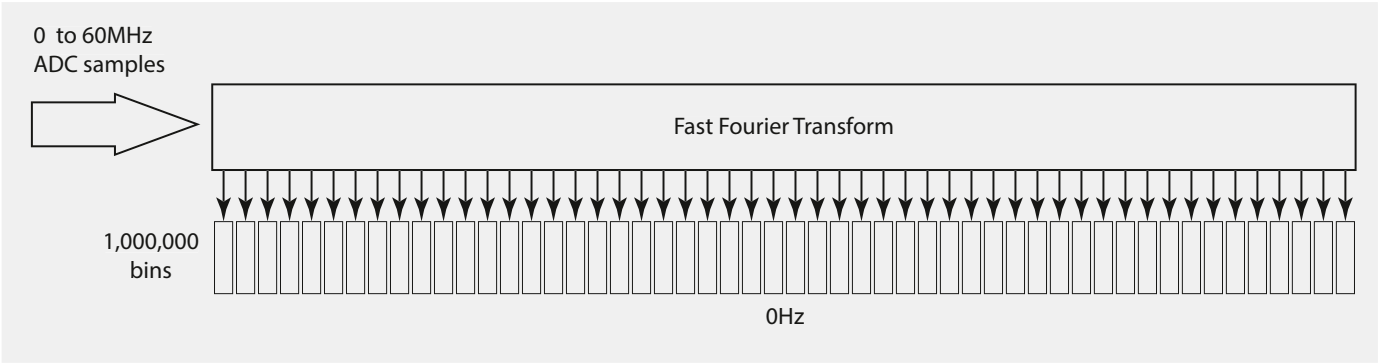

Fig. 4: Illustration of a Fast Fourier Transform output.

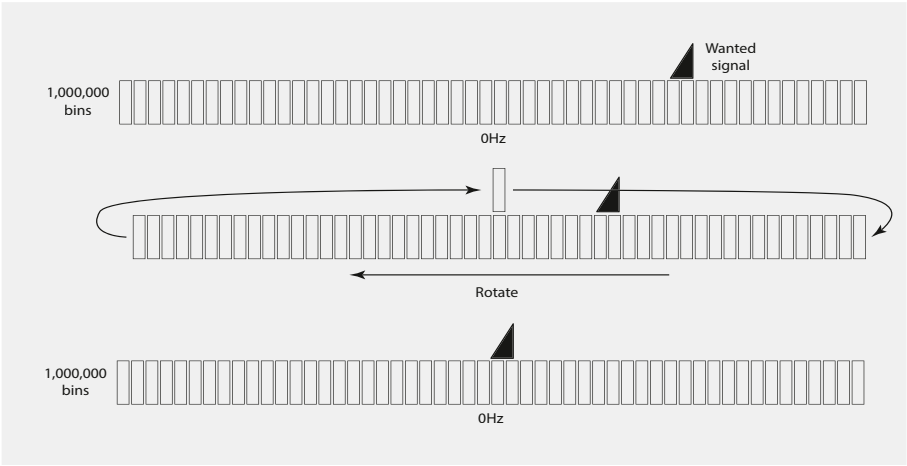

Fig. 5: Frequency conversion by rotating FFT bins.

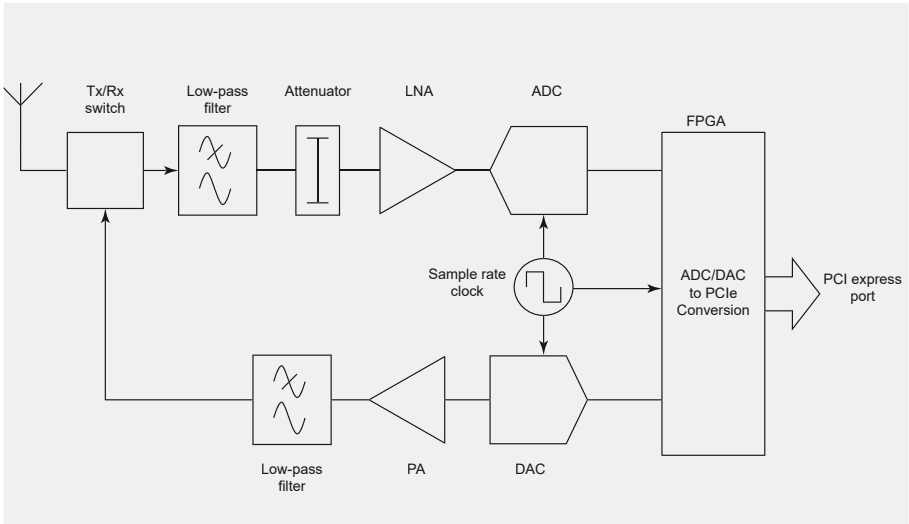

Fig. 6: Receiver hardware architecture for Minerva SDR.

In many SDR architectures it is common to swap to the frequency domain to do some filtering or similar and then swap back to the time domain. This process tends to be very processor hungry, especially if you want to change the entire captured spectrum. Phil's proposal is to run one big conversion to the frequency domain at the beginning of the process and then do as much processing as possible in that domain, only changing back to the time domain much further down the processing chain. This master FFT can be stored in memory and used to create multiple receivers.

This is a very neat solution but there have been several technical problems to overcome to make it viable. The first was a processing problem because it's not possible to continuously process/store all the incoming ADC data. The answer was to process the incoming data stream in overlapping blocks. Although this causes a few aliasing problems, these are easily corrected later in the process. Building the master FFT for the entire frequency range of the SDR demands a lot of FFT points and the example case used a 122.88MHz sample rate and required a million-point FFT. Having built the master FFT, the next problem was how to frequency shift the wanted signal down to baseband or 0Hz, which is in the centre of the FFT. The solution was to simply to rotate the FFT data bins, **Fig. 5**. You can only rotate in exact bin sizes so if the bin size was 3kHz you have to rotate the FFT in 3kHz steps. This translates to only being able to tune in 3kHz steps but this is another problem that has a solution later in the process. There is still a lot of data to process, but this is being handled by utilising the PC's graphics card.

The evolution of fast-moving 3D gaming environments has driven the availability of low-cost, powerful graphics card that are packed with parallel processing cores. It is not uncommon to have over 200 cores in the current generation of cards. These cores are ideal for processing large FFTs. The HPSDR team have been working hard on this project because it enables the use of much simpler and cheaper transceiver hardware. I've shown the new DFC hardware architecture in **Fig. 6**. Although it still includes an FPGA, this is only required to transfer the data from the ADC/DAC to the PCIe bus. This functionality is a standard feature in FPGAs so it only requires a few lines of code in a smaller and cheaper device. This new architecture is being progressed by the HPSDR team under the project name Minerva. The development is well advanced and the team are currently experimenting with prototype boards. If you'd like to understand more about the challenges of this new architecture, I recommend you watch Phil's excellent video.